ERD

Entity Relationship Diagram

Definition

An ERD, or Entity Relationship Diagram, is a structural diagram that, like the name indicate, use to visualize the relationships between entities. Mostly used in database design, the model's capable to highlighting two most important information: the entities within the system and the relationships between these entities.

Components

To demonstrate the idea of the database, ERD focused, on three elements:

- Entity: The thing that have data stored. Example: Cat, Dog,...
- Relationship: How entities related to each other. Example: Library can have one to many books.
- Attribute: Properties of the entity. Example: Color, Gender,...

Relationships

There are many relationship types in ERD, in fact, ‘one to many’ and ‘many to one’ can be considered to be two different types of relationships. Thought we don’t need to understand each of them, to understand how relationship in ERD work. Instead, when deciding the relationship, between entities, focus on two question:

- Are they optional?
- What is the required number from each side?

Relationship - Optionality

Optionality within an ERD indicate the dependence of one entity on another. In an ERD, this can also demonstrate as zero or one, optional or mandatory. Take the example of human and bank account relationship, a human can have a bank account, but doesn't mean that they need a bank account to be considered legit human, so they can have zero bank account. We can say that, human has optional relationship with bank account.

On the other hand, bank account need someone to have someone registered, to consider to be legit. So we can say that, it have mandatory relationship with human.

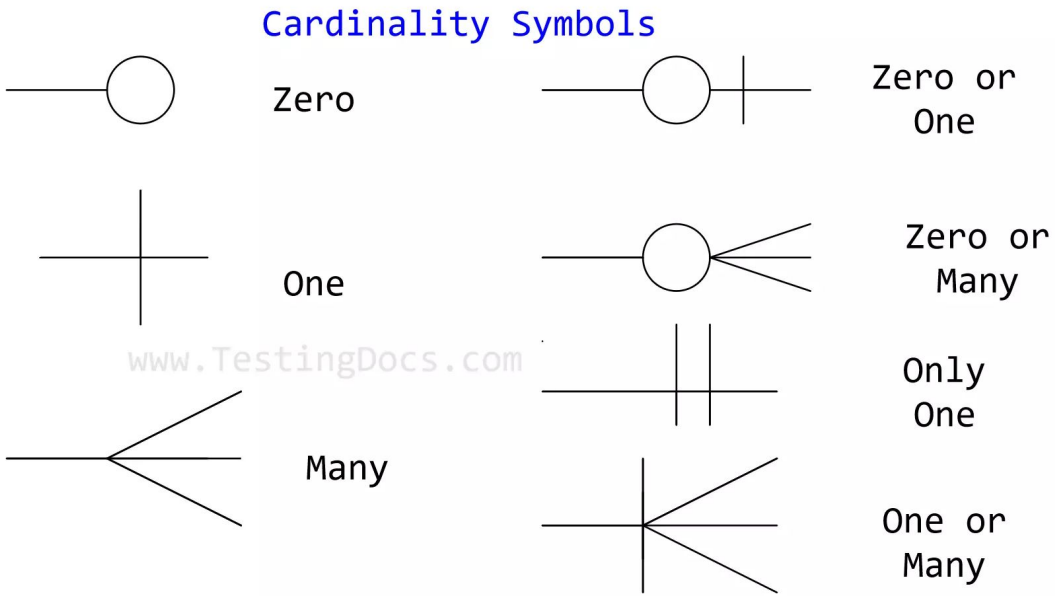
Relationships - numbers

The next question pertains to the number relationships between entities. Fortunately, we don't need to deal with exactly number here, so no need to deal with 2, 3, 4, 5, instead, there is only one question we should focus on. Literally, whether it one or more than one. We can also call this one or many question.

If we look at these relationship from both sides we can see that they can summary into three main types:

- One to One: The two entity can only have one each of another.
- One to Many: One entity can have more than one instant of other entity.
- Many to Many: Both of entity can have more than one of another.

Symbols

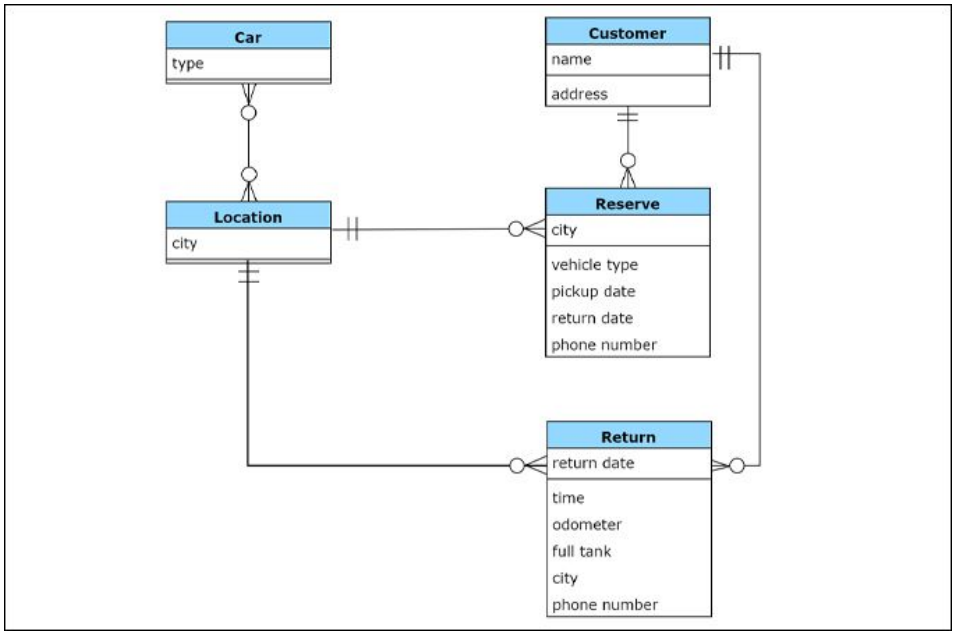

Cardinality Symbols
Zero
Zero or One
One
Zero or Many
www.TestingDocs.com
Only One
Many
One or Many

Symbols explanation

On the left are the basic symbols, the building blocks for the whole relationship system. The circle one indicate optional or zero. The second one, on the other hand, can be used in both number question and optional question, and in both case it can be understand at ‘one’. The last one symbolize ‘many’.

Using that we can explain the four relationships on the right side. The first two, with the zero symbol, can be understand as optional, combine with other parts can be understand as optional one and optional many. Applying the same rule, we can see that the two bellow can be understand as mandatory one and mandatory many.

Sample



As we can see from the sample, the data structure is separated into entities that contain attributes.
And all of the identities is connected through the relationship lines.

Key attribute

One of the most important concepts when it comes to ERD, is the concept of Key attribute. While relationship aims to demonstrate how each identity relates to each other, key attribute aims to demonstrate how the entities are connected to each other. This includes:

- Primary key: A identifier of an entity. It can't be null.
- Foreign key: A key attribute that works as a link from one entity to another.
- Super key: Serves the same purpose as primary key, but isn't limit to one per table, and can be used on set of attributes.
- Candidate key: A minimal set of Super key.

Normalization

In ERD, normalization is the process of organizing data to reduce redundancy and dependency inside of the database. This become achievable through the application of frameworks such as:

- First Normal Form: Each table must have primary key that contain primary key which identify each record. Additionally, this form involve removing any repeating groups or array, by move them to different table.
- Second Normal Form: Each non-key attribute in the entity should depend on the primary key, which mean that any attributes which are only partially dependent on the primary key will be moved to another table.
- Third Normal Form: Demand every attributes that are not dependent on the primary key to be moved to different table.

Some can say each number of form is one step further in classified the data inside of the database.

Denormalization

The strategy to decrease the impact of normalization in exchange for increased query performance. The name come from the fact that this technique involves adding redundant data or grouping data, which can be considered the opposite to normalization.

In someway, the two techniques balance each other out. By reducing the number of tables needed to be accessed, denormalization offer several benefits, such as performance improvement, simplified data access and reduced complexity.

Summary

Some would argue that an Entity-Relationship Diagram is one of the best tools available for working with databases, and the information provided in the slide above illustrates why. Thanks to the fact that components are clearly defined, ERD allows the structure of the database to be presented in a readable format for humans.

By using ERD as a basis, it becomes much easier to identify requirements. Additionally, actions such as normalization and performance optimization become possible. These are just the tip of the iceberg when it comes to what ERD can offer. Factors like scalability, maintenance, and tool integration can also be improved using ERD.

All of these reasons make ERD an essential part of database design.

OOP

Object Oriented Programming Paradigm

Definition

Object-Oriented Programing, or OOP, like the name suggests, is a type of programming paradigm that uses the concept of object for data manipulation. It can be said that the entire of OOP is built around the concept of object. Therefore, to understand it, we should trace back how objects come to be.

An object is an instance of a class, and class's components can be simplified into attributes and methods. Therefore to understand OOP, we need to understand: object, class, attribute, and method.

- Object: A instance of class, created for data manipulation.
- Class: A frame of a concept.
- Attribute: Variable of the class.
- Method: Function of the class.

Sample

For the sake of easy demonstration, let make a sample of a class called “animal”. This class has attribute name and method call. By using this we can create as much animal as we want. However that also mean that, we can use class, after all, if we use class by itself, when it change, all animal will change. And that is why we need to use object.

An object is a instance of class, so even if we change it, all other object won’t be affected. This mean that we don’t need to repeatedly use the same code over and over again for a slightly different variation.

Four pillars of OOP

Looking at how OOP has become a fundamental part of every modern programming languages, it's hard to imagine the world without it. However thing were not away this way. When the concept of programming language first emerged, POP, or Procedural Oriented Programming language, is the most famous approach.

Nevertheless, it doesn't take long for OOP to ascend to the throne, all of this was thank to the undeniable benefits brought about by the four pillars of OOP:

- Encapsulation.
- Abstract.
- Inheritance.
- Polymorphism.

Encapsulation

There is many way to understand this term, but let's not make unnecessarily complicate it. Encapsulation, mean containment, and in OOP it do exactly that. It contains all the importance information inside of the object, allowing program to selectively expose the information to the outside world.

Encapsulation typically appear in OOP languages, in a form of variable state like public, private and protected.

Utilizing this can prevent the object from acting outside of it intended purpose, adding security element into the structure level of the code.

Abstraction

This pillar can be understand as the ability to simplify complex system into a smaller and more manageable parts. Achievable through the act of defining abstract class, which work as a blueprint for other part of follow.

Normally in most of OOP language, this concept is applied through the use of interfaces.

Applying this principle can highly increase the readability of the code, and at the same time, increasing the code reusable and organization. Since the data is hidden, it can highly increase the security of the code.

Inheritance

In some way, Inheritance is the most powerful capacity. Imagine you have a class animal, but can't use it and instead, need some more specific classes like dog or cat. Thing is, the dog and cat is also animal, so they has a lot of similarity to the class animal. Normally it would mean that, you have to rewrite the class, but with inheritance it doesn't have to be the case.

Inside of most OOP languages, all you need to do is inherit from the class animal, and the class dog and cat will have everything the animal class has.

There is no need to say how much coding can be saved using inheritance. In reality, cat and dog doesn't have to be the limit, there can be rat, rabbit, raven. And the more there is the benefit become more notable.

Polymorphism

Inheritance can be powerful, but it alone is not perfect, for instant let say that the animal class has a method ‘call.’ Assume that all animal have a method call, but every animal has a different call, for example dog is “bark” and cat is “meow.” Since their call are not the same as inheritance, does that mean they have to rewrite the whole class? This is where polymorphism step in.

Take C# for example, there is a modifier called ‘virtual’, if the parent class has this, then the children class can change the method through the modifier called ‘overrride.’ This mean there is no need to rewrite the whole thing.

Some can say that polymorphism perfected inheritance.

Summary

Considering all the benefits of the four pillars, it is easy to see why OOP effectively replaced POP and became the most prevalent programming paradigm. It is simply far more efficient as a programming approach.

As time passes and programs continue to grow in size, recording everything every time you want a change is virtually impossible. On the other hand, OOP allows you to break the code into classes. When the system needs to perform an action, all it requires is to call a method from the object. Compared to rewriting the same code thousands of times, letting the system call an object thousands of times is a much easier and more scalable approach.

All of these factors make OOP the paradigm of the future.